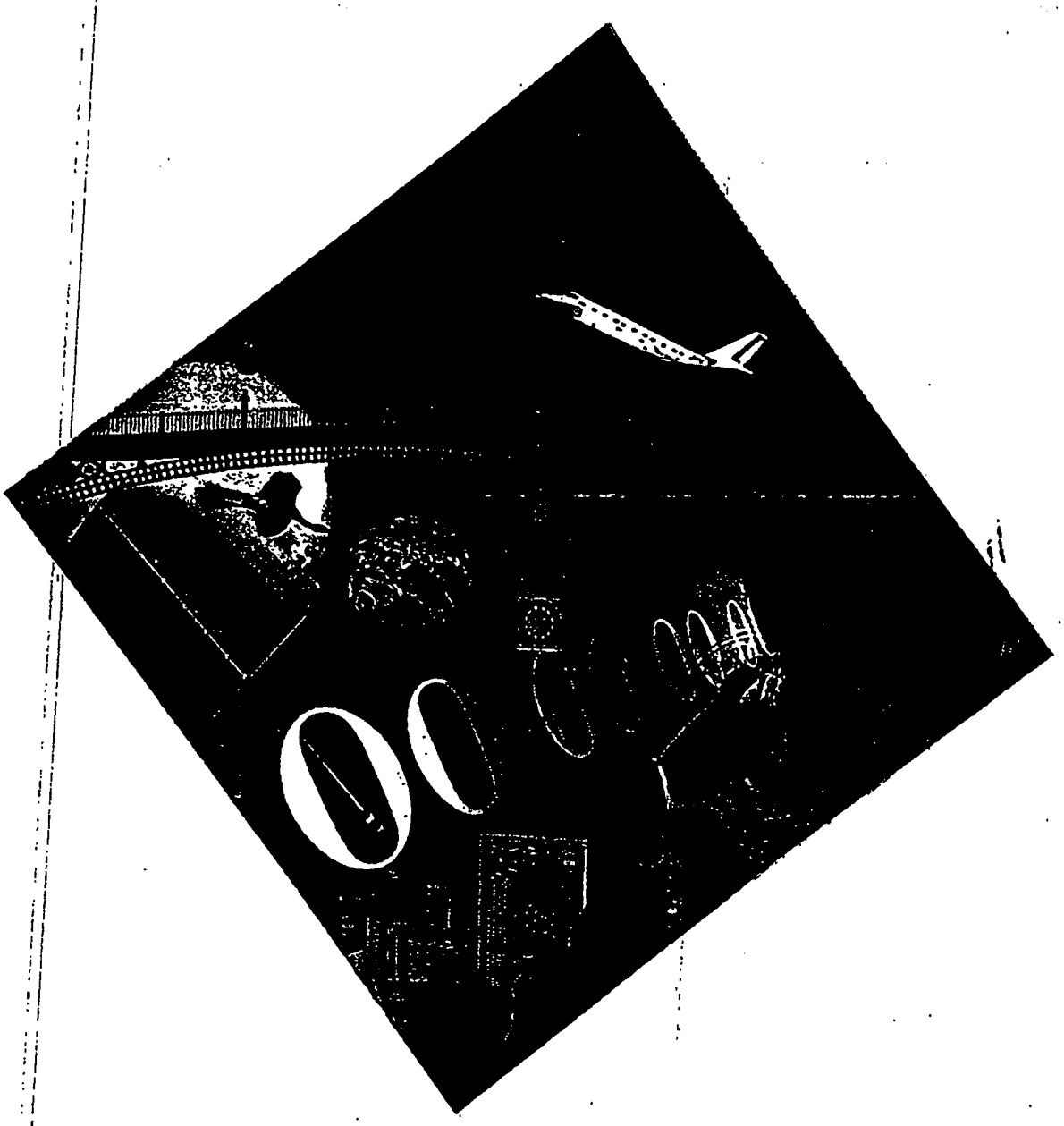
Third Edition

IBM

The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation



96

GC28-1251-02

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xv.

Third Edition, May 1996

This is a major revision of, and obsoletes, GC28-1251-01.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@vnet.ibm.com
World Wide Web: http://www.s390.ibm.com/

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Chapter 4. Reformatting Year-Date Notation

This chapter provides a number of techniques that you can employ to correct improper date notation and use. Because some techniques are appropriate only to unique situations, this section also lists the advantages, disadvantages, and IBM recommendations for their use.

When selecting a proposed Year2000 solution, evaluate the following factors:

1. What is the external impact due to incompatible date format changes?

   That is, what other programs or what output will be affected and to what extent will those programs require change if this solution is implemented for this particular program?

2. How current are the program modules that reference the date formats externalized by the exposures?

   That is, are there any plans to either eliminate or replace this particular program or routine, the programs that input to it, or those that receive or use its output?

3. What functions will be impaired due to Year2000 exposures?

   That is, will any mission-critical function within your company be compromised due to not reworking or replacing a particular program?

Solutions and Techniques

As you identify Year2000 exposures by the approaches described in Chapter 3, "Identifying 2-Digit-Year Exposures" on page 3-1, your next step is to rework the current program and data exposures to make your applications Year2000-ready. You can apply the following solutions to remove potential Year2000 exposures. Each solution is presented with an example technique to change the potential exposure. These suggested techniques require both program and data changes. Several solutions and techniques and their associated pros and cons follow:

Solution #1: Conversion to Full 4-Digit-Year Format

This solution is a 4-digit solution that externalizes a 4-digit-year format.

This approach requires changes to both the data and the programs by **converting all references and/or uses** of 2-digit-year format (YY) to 4-digit-year format (YYYY). It also requires that you convert all software programs that reference or use the updated data simultaneously, or use a 'bridging' mechanism to perform the conversion between old and new data and programs. Refer to "Bridge Programs Help Stage Format Conversions" on page 4-10 for more details on bridge programs. (You can accomplish this program and data conversion in steps.) Otherwise, you will immediately encounter data integrity problems caused by the inconsistency of date/time data formats.

To ease your migration, you might consider ignoring any non-impact (cosmetic) data fields in the YY format. A cosmetic date is one, that if externalized, is only interpreted by humans. Such occurrences might include the date on an output separator page or a display-only date on a screen in a panel-driven application.

Note: Be careful when selecting those situations that you decide to ignore and call cosmetic only. Be certain that they will not cause any data integrity exposures or ambiguity or are not accessed by any other program. Such instances of non-problem YY formats appear in a report header that shows the printing date of the report. The date is meant for human understanding only, not computer program manipulation. Consider the potential for future change. For example:

- Today's reports might be written to a data set tomorrow
- Display-only dates today may prove useful as a collating value when archiving that output tomorrow to meet a new business or government standard.
- Even when viewed by a human, 2-digit dates can prove ambiguous if the data spans 100 years.

If you allow the end user to continue to input 2-digit dates for compatibility and ease of data entry, then the responsibility to translate that data into a full 4-digit date falls to you, the application or systems programmer. One possible solution is to apply a context-sensitive prompt to allow the user to select a century indicator. For example, allow all dates to be entered as 2-digit dates and automatically prefix those with the current century unless the date is a future date or historical date. What constitutes "future" or "historical" is your decision but could be any date other than today's current day, week, month, year, and so on. Using this scheme, a future date in context of a loan maturity date could be set to 20yy, or a historical date automatically forces the user to select a century from a 'choose a century' (...16, 17, 18) prompt list.

Pros and Cons

Pros

1. Can provide 4-digit-year format. It is considered to be the only complete, permanent, and obvious solution.
2. Provides increased security against potential inappropriate decisions today if you do not selectively ignore 'cosmetic-only' situations.
3. Can ease your migration if you selectively ignore 'cosmetic-only' situations.

Cons

1. Need to convert the year data from 2-digit format to 4-digit format in all cases.
2. Requires that you relocate adjacent fields in the date field layout, and usually requires that you increase record lengths.
3. Inherent future risk in initial assessment that determined a particularly situation can be ignored as 'cosmetic only'.
4. Increased DASD space usage required due to data field expansion of data (consider including not only active but also archive data) and duplicate DASD space during conversion.
5. Might experience a performance impact due to increased time in processing and date access.
6. Some programming languages allow integer dates that are offset from a base date to be stored in files, data bases or passed as parameters between programs. Such integer dates provided by COBOL intrinsic functions, Language Environment callable services,

the CICS FORMATTIME command DAYCOUNT option, and other similar functions must conform to the standard YYYYMMDD format. This standard eliminates potential ambiguous data and errors due to each integer-date system using a unique starting date. Therefore, the potential for mixing incompatible integer dates when passed outside a single source module is extremely high and must be avoided.

Solution #2: Windowing Techniques

This is a 2-digit solution that externalizes either 2-digit or 4-digit-year formats. This approach requires changes to your programs only; no data changes are required.

CAUTION:
These approaches can be applied only to dates within a maximum 100-year period at any one time. This solution is considered temporary because there is no guarantee that in the future, your applications will not expand to process dates that are more than 100 years apart. Therefore, this approach always carries with it a potential future exposure. (For example, humans are living longer. Therefore data bases that include birthdays (medical, civil, insurance, and so on) and the applications that access that data are already at risk with many dates spanning 100+ years.)

Two types of windowing techniques have been defined: the fixed window technique and the sliding (rolling) window technique.

Fixed Window Technique

The fixed window technique uses a static 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to a specific year within the 100-year interval.

Consider this specific example: if the years of date-related data of your application fall in the range of 1 January 1960 to 31 December 2059, you can use a 2-digit year to distinguish dates prior to the year 2000 from the year 2000 and beyond. If using the current system year of 1995, the number of years in the past and future are specified as 35 and 64, respectively. Program logic determines the century based on the following data checking. If the 2-digit year representation of a specific year is xy then if:

- xy≥60, then it is a 20th century date (19xy)
- Otherwise (that is, xy≤59), it is a 21st century date (20xy).

If, for example, you need to maintain a window of 35 past years and 64 future years, such that next year, 1996, your application can successfully deal with dates in the range 1961 through 2060, you need to adjust this program checking every year. The inherent future risk when employing this technique is obvious, and when compared to the sliding window technique is far less desirable.

Pros and Cons

Pros

1. No need to expand the 2-digit-year data to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).
4. Can be useful if the particular program is being phased out, and a temporary solution is appropriate.

Cons

1. Potential exposures exist when/if the function of the software application needs to process years beyond the range of 100 years.
2. Expect a performance impact in direct proportion to the quantity of date processing the particular application handles due to the overhead of 2- to 4-digit-year conversion.
3. All programs that use the fixed window technique may need to be manually updated on a yearly basis depending on how your date routine is packaged.
4. All programs that accept output from the fixed window technique must use the same assumptions (current date, past and future windows).
5. Retaining a 2-digit year representation does not provide collating sequence support. Nor does the use of a fixed window technique provide indexing sequence support when 2-digit years are used as index keys in indexed files. You will need to provide additional processing to obtain correct collating and indexing sequence output.

Sliding Window Technique

The **sliding window** technique uses a self-advancing 100-year interval that generally crosses a century boundary. This technique determines the century of a 2-digit year by comparing the 2-digit year against a window of 100 years. The user specifies the number of years in the past and future relative to the system year (generally the current year) that the system sets[1] and maintains. Your applications can access the date that the system sets and automatically advances. This is the main advantage of using a sliding window over the fixed window (where the window is immovable without manually revising the programs each year).

As appropriate to your application environment, you can maintain more than one window. For example, you could set one window to process historical dates, one for mortgage dates, one for birth dates, and so on; and the program adjusts the system date and past and future windows to meet the specific application's needs.

Consider this specific example. If the dates in your application fall into a range of 35 years in the past and 64 years into the future, based on the current year, 1995,

---

[1] IBM product, Language Environment, provides an option whereby you can set the system year to other than the current year. This flexibility then allows you to set a 100-year range you require; it need not even contain the current year. Refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1.

your program can accept and accurately deal with dates of 1960 through 2059. Next year, 1996, the window advances and your application accurately deals with dates of 1961 through 2060.

Graphically, Figure 4-1 on page 4-5 illustrates this example using the current (1995) 100-year window and that same window when the current system date has progressed to the year 2024.

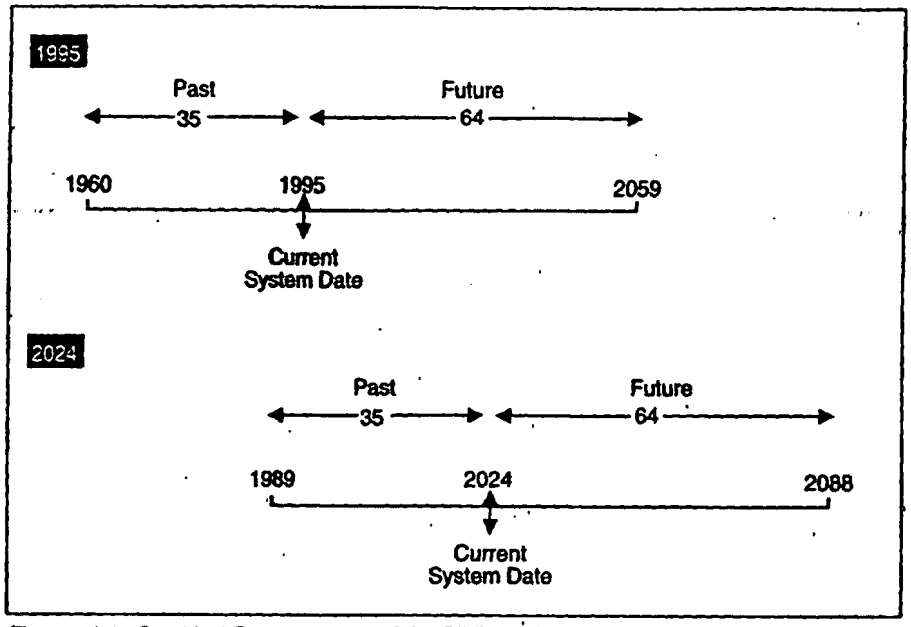


Figure 4-1. Graphical Representation of the Sliding Window Technique. (Using two current system dates, 1995 and 2024, as an example.)

A sliding window approach requires programming logic to interpret the meaning of all 2-digit year data. Such additional programming logic could be packaged into a common data/time service routine, callable from a 2-digit year data exploiter. This would reduce the programming overhead and impact to the calling programs. IBM product, Language Environment, provides common date/time service routines with sliding window features. By default, Language Environment uses a window of 80 years in the past and 20 years into the future that automatically adjusts based on the current year date. For details on using and setting the past/future window range, refer to Chapter 7, "Tool Categories and Available Tools to Ease Year2000 Changes" on page 7-1. For an example of how DFSORT/MVS is implementing a sliding window to sort, merge, and transform 2-digit year data to 4-digit year data, refer to "DFSORT" on page 7-23

Pros and Cons

Pros

1. No need to expand the 2-digit-year format to a 4-digit format.
2. Can provide 4-digit-year format for data reference.
3. Can distinguish years from different centuries using only 2-digit-year format (provided the years being processed are in the range of 100 years at any one time).

WITT automatically records and plays back all keystroke and mouse movements and compares and identifies test inconsistencies between benchmark test cases and test cases run after enhancements and fixes are made to a program. WITT also allows scripting in 2/REXX to add program intelligence to the test cases. This makes WITT test cases easy to update, maintain, and reuse for future testing.

WITT/OS2 installs on an OS/2 desktop and allows the testing of MVS, VM, IMS, CICS, OS/400, and OS/2 PM and Text (ie, Micro Focus, CICS/OS2) applications. WITT/PM installs on an OS/2 desktop and allows the testing of only OS/2 PM and Text applications. X/WITT installs on an AIX desktop and allows the testing of AIX X Window applications, as well as remote client applications in X Windows environments on SUN, HP, APOLLO, and MVS. WITT/Windows installs on a Windows 3.1 desktop and allows the testing of Windows workstation applications.

SORT

DFSORT V1R13 will enhance its Year2000 capabilities by providing the ability to sort, merge, and transform 2-digit years according to a specified sliding or fixed century window. New Y2C, Y2Z, Y2P, and Y2D formats, in conjunction with a new Y2PAST installation and run-time option, allow you to handle 2-digit year data in the following ways:

- Set the appropriate century window for your applications. For example, set a century window of 1915-2014 or 1950-2049.
- Order 2-digit character, zoned decimal, packed decimal, or decimal year data, according to the century window, using DFSORT's SORT and MERGE control statements. For example, order 96 (representing 1996) before 00 (representing 2000) in ascending sequence, or order 00 before 96 in descending sequence.
- Transform 2-digit character, zoned decimal, packed decimal, or decimal year data to 4-digit character year data, according to the century window, using DFSORT's OUTFIL control statement. For example, transform 99 to 1999 and 04 to 2004.

These DFSORT enhancements allow you to continue to use 2-digit years for sorting and merging, and assist those situations when you want to change 2-digit-year data to 4-digit-year data.

Additional information about DFSORT/MVS and its year2000 enhancements is available on the World Wide Web at URL:

```
http://www.storage.ibm.com/storage/software/sort/srtmhome.htm
```

Sliding Century Window

A new installation and run-time option allows you to specify a sliding or fixed century window to be used with 2-digit years. Y2PAST=s specifies a **sliding** century window starting s years before the current year. For example, if the current year is 1996, Y2PAST=80 starts the century window at 1996 - 80 = 1916, providing a century window of 1916 through 2015. In 1997, this century window automatically slides to 1917 through 2016.

Y2KPAST=f specifies a **fixed** century window starting at f. For example, Y2KPAST=1950 starts the century window at 1950, providing a century window of 1950 through 2049. Thus, Y2PAST allows you to control how DFSORT interprets the 2-digit years 00-99 on a site-wide or application-specific basis.

As an example, both Y2KPAST=1915 and Y2PAST=81 used in 1996 give a century window of 1915 through 2014, and result in the following interpretation of 2-digit year formatted data by DFSORT:

yy	Interpreted as:
00	2000
14	2014
15	1915
61	1961
62	1962
99	1999

2-Digit Year Formats

New formats allow you to identify 2-digit character, zoned decimal, packed decimal and decimal year data for special DFSORT processing as follows: (yy represents 2-digit year data in the examples below.)

Format	Meaning
Y2C	identifies 2-digit, 2-byte character year data such as C'yy', C'mm/dd/yy', or C'yy.mm.dd'
Y2Z	identifies 2-digit, 2-byte zoned decimal year data such as Z'yy', Z'mmddyy', or Z'yymmdd'
Y2P	identifies 2-digit, 2-byte packed decimal year data such as P'yy', P'dddyy', or P'yymmdd'
Y2D	identifies 2-digit, 1-byte decimal year data such as X'yy' or P'yyddd'

Sorting and Merging 2-Digit Years

You can use the new Y2C, Y2Z, Y2P, and Y2D formats in DFSORT's SORT and MERGE statements to identify specific 2-digit year data to be ordered according to the century window.

A simple example of the control statements to sort a C'mm/dd/yy' field (assume the current year is 1996) follows:

```
* Set the century window to 1962 through 2061
  OPTION Y2PAST=34
* Sort C'mm/dd/yy' as C'yymmdd'
  SORT FIELDS=(7,2,Y2C,A, * sort yy using century window
               1,2,CH,A,   * sort mm
               4,2,CH,A)   * sort dd
```

These control statements provide the following sort results:

Input Data (CH)	Sorted Output Data (CH)
06/22/15	03/18/62
10/03/00	09/01/99
11/14/61	10/03/00
08/16/14	08/16/14

09/01/99	08/17/14
03/18/62	06/22/15
08/17/14	11/14/61

Transforming 2-Digit Years to 4-Digit Years

You can use the new Y2C, Y2Z, Y2P, and Y2D formats in the OUTREC operand of DFSORT's OUTFIL statement to identify 2-digit year data to be changed to 4-digit year data according to the century window.

A simple example of the control statements to transform a P'yyddd' field follows:

```
* Set the century window to 1970 through 2069
  OPTION COPY,Y2PAST=1970
* Change P'yyddd' to C'yyyy/ddd'
 OUTFIL FNAMES=Y4,
  OUTREC=(1,1,Y2D,      * change X'yy' to C'yyyy' using
*                       *   century window
         C'/',          * insert C'/'
         2,2,PD,M11)    * change P'ddd' to C'ddd'
```

This code provides the following transformation results:

Input Data (HEX)	Transformed Output Data (CH)
92012F	1992/012
70225C	1970/225
69153F	2069/153
00001F	2000/001
99321F	1999/321
12054C	2012/054

COMUDAS (COMmon Uithoorn DAte Services)

COMUDAS (program product # 5788-HBB) is a common date routine that has been developed to replace all existing date routines, used by the IBM Uithoorn Lab, The Netherlands. COMUDAS offers all necessary functions for validation, conversion, and calculation of dates in any format. A standard interface must be used to call the date routine's load module dynamically.

The package also offers the possibility to use separate functions by means of NCAL's, that can be linked statically. This might be useful when converting large files or databases with validated data into other formats (for example, when reformatting year-date notation in an application).

Functions are available for updating the Calendar Tables by means of the Online Facility, that also can be used to test the Date Routines, and to print calendars.

A CICS version, as well as an MVS version, of this package is available.